VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Komponenta výukového serveru TI - Asymptotická notace

Component of Educational Server for Theoretical CS - Asymptotic Notation

2017

Martin Frýdl

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Martin Frýdl**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Komponenta výukového serveru TI - Asymptotická notace

Component of Educational Server for Theoretical CS - Asymptotic Notation

Jazyk vypracování: čeština

Zásady pro vypracování:

V rámci diplomových a bakalařských prací vzniká výukový server pro předměty teoretické informatiky. Jedná se o sadu dynamických webových stránek umožňujících studentům pochopení různých typů úloh a problémů tím, že si mohou zadat na stránce libovolné zadání a zobrazí se jim řešení včetně postupu.

Cílem této bakalářské práce je vytvořit komponentu pro výuku asymptotické notace.

1. Vytvořte dynamické webové stránky umožňující uživateli následující:
a) zadat si funkce, podporovány budou nejméně všechny polynomiální funkce, logaritmy, exponenciální funkce, faktoriál,
b) zobrazit si porovnání zadaných funkcí pomocí asymptotické notace,
c) zobrazit si grafy jednotlivých funkcí včetně jejich konstantních násobků tak, aby z grafu bylo vidět porovnání asymptotické rychlosti růstu.
2. Dále vytvořte testovací stránku pro ověření získaných znalostí. Náhodně bude vybráno několik funkcí a uživatel bude vybírat, které vztahy vyjádřené asymptotickou notací mezi nimi platí.
3. Použité technologie budou voleny tak, aby byly stránky na straně klienta co nejméně platformně závislé.

Seznam doporučené odborné literatury:

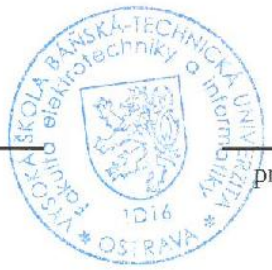
[1] P. Jančar: Teoretická informatika, VŠB-TU Ostrava 2007 (2010)
[2] M. Sipser: Introduction to the Theory of Computation, Thomson Course Technology, 2006

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Ing. Martin Kot, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 14.07.2017



doc. Ing. Jan Platoš, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 14. července 2017 ..... Frýdl .....

Rád bych poděkoval vedoucímu své bakalářské práce panu Ing. Martinu Kotovi, Ph.D za odborné vedení, cenné rady, věcné připomínky a čas, který mi věnoval při zpracování této práce.

Abstrakt

Tato bakalářská práce popisuje komponentu výukového serveru TI pro podporu výuky v oblasti asymptotické notace. Výsledná webová aplikace obsahuje nejen nezbytnou teorii a řešené příklady k dané oblasti, ale především nabízí uživateli možnost zadat vlastní funkce k porovnání pomocí asymptotické notace a ověření jeho znalostí této problematiky pomocí generovaných testů. Samotný text této práce se pak věnuje nejen popisu nezbytné teorie, analýze požadavků, návrhu aplikace a popisu použitých technologií, ale zejména pak samotné implementaci řešení.

Klíčová slova: asymptotická notace, teoretická informatika, webová aplikace

Abstract

This bachelor's thesis describes a component of an educational server for theoretical computer science to support teaching of this topic. In addition to providing the necessary theoretical background and solved exercises, this web application also allows the user to specify custom functions and analyse their asymptotic complexity. Another feature are automatically generated tests which can be employed to assess the level of user's knowledge of the subject. The written content of this work describes the necessary theoretical background, the analysis of the requirements, the design of this aplication, the information about the used technologies, and most importantly the description of the implementation of the solution.

Key Words: asymptotic notation, theoretical computer science, web application

Obsah

Seznam použitých zkratek a symbolů

API	–	Application Programming Interface
CS	–	Computer Science
CSS	–	Cascading Style Sheets
DOM	–	Document Object Model
JS	–	JavaScript
HTML	–	Hyper Text Markup Language
GUI	–	Graphical User Interface
TI	–	Teoretická Informatika
URL	–	Uniform Resource Locator

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Problematika Asymptotické notace je oblast teoretické informatiky, která je velmi důležitá při řešení úloh pomocí výpočetní techniky. Složitost zvoleného algoritmu má přímý vliv na to, jak bude daný algoritmus efektivní. Proto musíme mít nástroj, kterým dokážeme porovnat efektivitu a rychlost vykonávání jednotlivých algoritmů v závislosti na změně velikosti vstupních dat.

Cílem této bakalářské práce je navrhnout a implementovat webovou aplikaci pro podporu výuky této oblasti, s jejíž pomocí si mohou studenti nastudovat nezbytnou teorii a postup výpočtů prostřednictvím řešených příkladů. Také si mohou zadat vlastní funkce k porovnání, kde výstupem je nejen vyhodnocení, která ze zadaných funkcí je náročnější na vykonání, ale především graf těchto dvou funkcí, ve kterém si pomocí volených parametrů mohou zobrazit jejich průběh. V neposlední řadě pak tato webová aplikace nabízí ověření znalostí uživatele v podobě generovaných testů pro porovnání funkcí pomocí asymptotické notace.

Bakalářská práce je rozdělena do několika kapitol. Kapitola 2 obsahuje základní teorii dané problematiky. V kapitole 3 se nachází stručné upřesnění požadavků kladených na systém. Kapitola 4 se věnuje rozboru stěžejní funkce aplikace, kterou je porovnávání asymptotické rychlosti růstu pro uživatelem zadané funkce. V kapitole 5 pak návrh webové aplikace. Kapitola 6 se věnuje použitým technologiím, frameworkům a knihovnám. V kapitole 7 je popsána implementace samotného řešení. Závěrem kapitola 8 popisuje nasazení hotového řešení.

V průběhu vývoje byla aplikace testována a pravidelně prezentována vedoucímu práce prostřednictvím serveru HomeL. Tedy konkrétně na adrese http://homel.vsb.cz/ fry0050/bp/.

2 Teorie

Asymptotickou notaci používáme k určení složitosti algoritmu, neboli k určení jak je daný algoritmus rychlý (kolik provede elementárních operací) vzhledem k množině vstupních dat aniž bychom museli přesně vyčíslovat všechny konstanty a složitost méně časově náročných částí algoritmů, které se vykonávají sekvenčně s těmi náročnějšími. V následujících kapitolách platí:

- c je kladné reálné číslo ($c \in R$ a $c > 0$)
- n_0 a n jsou přirozená čísla ($n_0 \in N$ a $n \in N$)

2.1 Notace

Notace, které používáme k popisu asymptotické doby běhu algoritmů, jsou definovány jako funkce jejichž domény jsou množiny přirozených čísel. Takové notace jsou vhodné pro popis nejhorší možné doby běhu funkcí.

2.1.1 Notace O

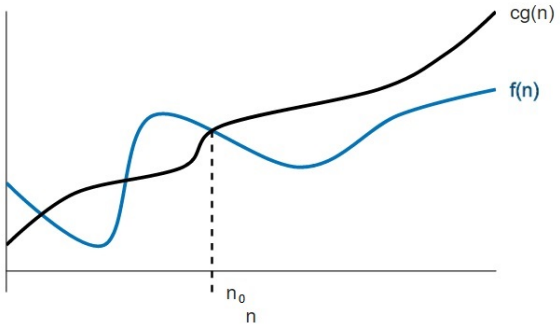
Notace O („velké O“) slouží pro omezení funkce shora v rámci konstantního činitele. Pro všechny hodnoty n napravo od nějaké konstanty n_0 je hodnota funkce $f(n)$ na nebo pod $cg(n)$ pro nějaké c.

Ukázku možného průběhu takovýchto funkcí můžeme vidět na obrázku 1.

Definice 1 *[1] Vezměme si libovolnou funkci $g : N \to R$. Pro funkci $f : N \to R$ platí:*

$$f \in O(g) \leftrightarrow (\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \leq cg(n)) \tag{1}$$

$O(f)$ je množina všech funkcí (popř. konkrétní funkce z množiny), které rostou **nejvýše** tak rychle jako f. Zapisujeme : $f \in O(g)$, popř. $f = O(g)$ pokud zápis $O(g)$ reprezentuje konkrétní funkci z množiny.



Obrázek 1: Graf funkce $f \in O(g)$

2.1.2 Notace Ω

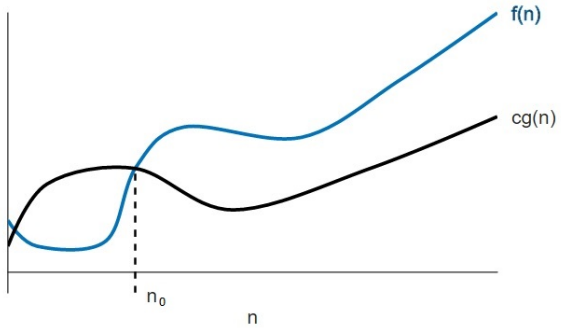
Notace Ω („velká Omega“) slouží pro omezení funkce zdola v rámci konstantního činitele. Pro všechny hodnoty n napravo od nějaké konstanty n_0 je hodnota funkce $f(n)$ na nebo nad $cg(n)$ pro nějaké c.

Ukázku možného průběhu takovýchto funkcí můžeme vidět na obrázku 2.

Definice 2 *[1] Vezměme si libovolnou funkci $g : N \to R$. Pro funkci $f : N \to R$ platí:*

$$f \in \Omega(g) \leftrightarrow (\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(cg(n) \leq f(n)) \tag{2}$$

$\Omega(f)$ je množina všech funkcí (popř. konkrétní funkce z množiny), které rostou **alespoň** tak rychle jako f. Zapisujeme: $f \in \Omega(g)$, popř. $f = \Omega(g)$ pokud zápis $\Omega(g)$ reprezentuje konkrétní funkci z množiny.



Obrázek 2: Graf funkce $f \in \Omega(g)$

2.1.3 Notace Θ

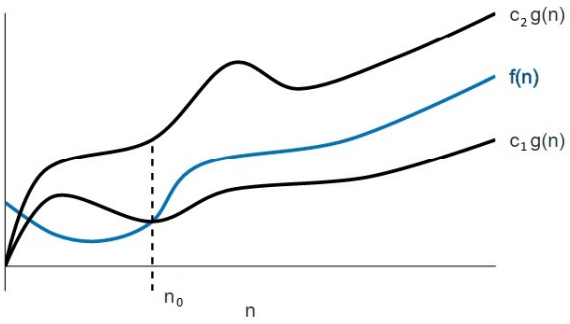
Notace Θ („velká Théta“) slouží pro omezení funkce zdola i shora v rámci konstantních činitelů. Pro všechny hodnoty n napravo od nějaké konstanty n_0 je hodnota funkce $f(n)$ na nebo nad $c_1 g(n)$ a zároveň na nebo pod $c_2 g(n)$ pro nějaké c.

Ukázku možného průběhu takovýchto funkcí můžeme vidět na obrázku 3.

Definice 3 *[1] Vezměme si libovolnou funkci $g : N \to R$. Pro funkci $f : N \to R$ platí:*

$$f \in \Theta(g) \leftrightarrow (\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0)(c_1 g(n) \leq f(n) \leq c_2 g(n)) \tag{3}$$

$\Theta(f)$ je množina všech funkcí (popř. konkrétní funkce z množiny), které rostou **stejně** tak rychle jako f. Zapisujeme: $f \in O(g)$, popř. $f = \Theta(g)$ pokud zápis $\Theta(g)$ reprezentuje konkrétní funkci z množiny.



Obrázek 3: Graf funkce $f \in \theta(g)$

2.1.4 Notace o

Definice O-notace a o-notace („malé o“) jsou podobné. Hlavní rozdílem je, že v $f(n) = O(g(n))$ platí mez $0 \leq f(n) \leq cg(n)$ pro nějakou konstantu $c > 0$, ale v $f(n) = o(g(n))$ platí mez $0 \leq f(n) < cg(n)$ pro všechny konstanty $c > 0$ [1] .

2.1.5 Notace ω

Obdobně jako o-notace pro O-notaci funguje ω-notace („malá omega“) pro Ω-notaci. Tuto notaci používáme pro označení dolní meze, která není asymptoticky těsná. Jednou z možných definic je: $f(n) \in \omega(g(n)) \leftrightarrow g(n) \in o(f(n))$ [1] .

2.2 Porovnávání funkcí

Mnoho z relačních vlastností reálných čísel můžeme aplikovat pro porovnání asymptotických funkcí. Říkáme, že $f(n)$ je asymptoticky menší než $g(n)$ pokud $f(n) = o(g(n))$ a $f(n)$ je asymptoticky větší než $g(n)$ pokud $f(n) = \omega(g(n))$ [1] .

2.2.1 Tranzitivita

$$f(n) = \Theta(g(n)) \wedge g(n) = \Theta(h(n)) \rightarrow f(n) = \Theta(h(n)) \tag{4}$$

$$f(n) = O(g(n)) \wedge g(n) = O(h(n)) \rightarrow f(n) = O(h(n)) \tag{5}$$

$$f(n) = \Omega(g(n)) \wedge g(n) = \Omega(h(n)) \rightarrow f(n) = \Omega(h(n)) \tag{6}$$

$$f(n) = o(g(n)) \wedge g(n) = o(h(n)) \rightarrow f(n) = o(h(n)) \tag{7}$$

$$f(n) = \omega(g(n)) \wedge g(n) = \omega(h(n)) \rightarrow f(n) = \omega(h(n)) \tag{8}$$

2.2.2 Reflexivita

$$f(n) = \Theta(f(n)) \tag{9}$$

$$f(n) = O(f(n)) \tag{10}$$

$$f(n) = \Omega(f(n)) \tag{11}$$

2.2.3 Symetrie

$$f(n) = \Theta(g(n)) \leftrightarrow g(n) = \Theta(f(n)) \tag{12}$$

2.2.4 Transponovaná symetrie

$$f(n) = O(g(n)) \leftrightarrow g(n) = \Omega(f(n)) \tag{13}$$

$$f(n) = o(g(n)) \leftrightarrow g(n) = \omega(f(n)) \tag{14}$$

2.3 Třídy složitosti

Ke klasifikaci algoritmů se obvykle používá tzv. asymptotická složitost, což je rozdělení algoritmů do tříd složitostí podle $O(n)$. Přitom platí, že algoritmus z vyšší třídy je pomalejší než algoritmus z předchozí třídy.

Jednoduše řečeno: pokud spadají dva algoritmy do různých tříd asymptotické složitosti, pak vždy existuje takové množství dat, od kterého je asymptoticky lepší algoritmus vždy rychlejší, bez ohledu na to, kolikrát je některý z počítačů výkonnější[2].

V tabulce 1 můžeme vidět několik nejvýznamnějších tříd složitosti.

Škála v nekonečnu slouží k rozlišení jednotlivých tříd. Říká, že pokud se n blíží k nekonečnu, tak neexistuje reálná konstanta taková, aby byl algoritmus z vyšší třídy rychlejší než ten z třídy předchozí[2].

Tato škála, která nám slouží k asymptotickému porovnání jednotlivých tříd, je zobrazena v poznámce 1

Tabulka 1: Nejvýznamnější třídy složitosti

Značení	Význam
$O(1)$	konstantní
$O(log(n))$	logaritmická
$O(n)$	lineární
$O(n \cdot log(n))$	lineárně logaritmická
$O(n^2)$	kvadratická
$O(n^3)$	kubická
obecně $O(n^k)$	polynomiální
obecně $O(k^n)$	exponenciální
$O(n!)$	faktoriálová

Poznámka 1

$$1 << log(n) << n << n \cdot log(n) << n^k << k^n << n! << n^n, \, kde \, k > 1 \tag{15}$$

3 Specifikace požadavků

Tato kapitola analyzuje požadavky na novou komponentu výukového serveru pro podporu výuky teoretické informatiky. V této analýze jsem vycházel nejen ze zadání bakalářské práce, ale také z průběžných požadavků vedoucího práce a svých vlastních zkušeností a předpokladů získaných zejména studiem předmětu Úvod do teoretické informatiky.

3.1 Důvod vzniku systému

Tato webová aplikace vznikla pro podporu výuky v oblasti teoretické informatiky, konkrétně problematiky asymptotické notace. Zejména tedy na porovnávání funkcí pomocí této notace. A sice pomocí vlastní kalkulačky, náhodně generovaných testů a výukou základní teorie s použitím ukázkových řešených příkladů.

3.2 Primární funkce

Primární funkcí systému je umožnit uživateli pochopit porovnávání funkcí pomocí asymptotické notace a to zejména prostřednictvím uživatelem zadaných funkcí a následným textovým a grafovým výstupem.

Uživateli je umožněno zadávaní nejen samostatných polynomiálních a exponenciálních funkcí, logaritmů, faktoriálů a konstant, ale také zadat operace násobení, umocňování a sčítání.

Rozbor této funkcionality nalezneme v následující kapitole 4.

3.3 Uživatelé

Systém je určen především pro studenty předmětů z oblasti teoretické informatiky, kteří se zajímají o problematiku asymptotické notace. Dále pak pro samotné pedagogy jako pomůcka pro výuku.

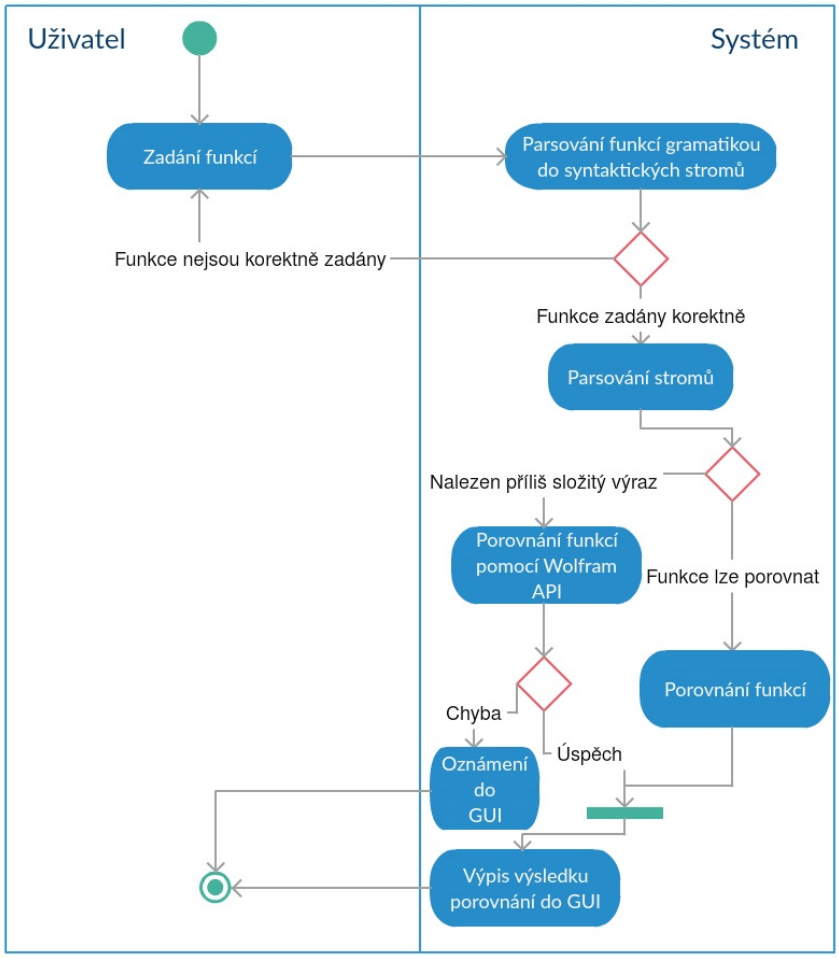
Systém může v případě nasazení webové aplikace na veřejný server využívat i široká veřejnost, neboť se jedná o standardní webovou aplikaci, na kterou není nutná žádná forma registrace.

4 Analýza primární funkce aplikace

Tato kapitola rozebírá řešení primární funkce aplikace, a sice porovnání asymptotické rychlosti růstu pro dvě uživatelem zadané funkce.

4.1 Princip porovnávání funkcí

Princip porovnávání funkcí je znázorněn diagramem na obrázku 4. Jednotlivé body znázorněné v diagramu budou podrobněji vysvětleny v následujících kapitolách.



Obrázek 4: Princip porovnávání funkcí

4.2 Bezkontextová gramatika

Uživatelem zadávané funkce procházejí validací pomocí bezkontextové gramatiky zobrazené v poznámce 2. Rekurzivní sestup touto gramatikou zároveň tvoří i syntaktický strom zadaných funkcí, který se dále používá pro další práci.

Poznámka 2 Bezkontextová gramatika pro zadávání funkcí

S → E <end>
E → TG
G → ϵ | +TG
T → FU
U → ϵ | *FU | FU
F → XY | lZ
X → n | K | (E)
Y → ^X | ! | ϵ
Z → nP | ogW
W → _KP | P
P → (R
R → n) | K)
C → 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
K → CL
L → CL | ϵ

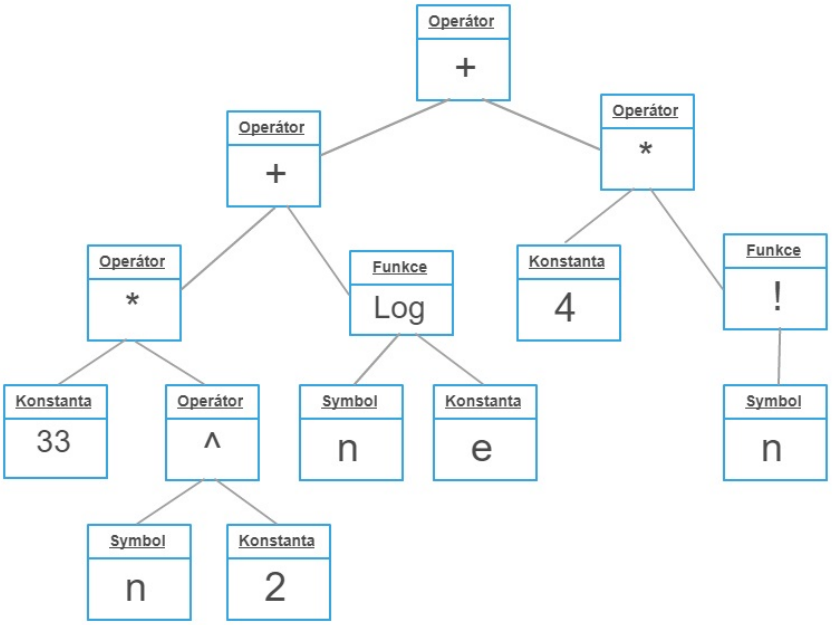
Z návrhu gramatiky můžeme vidět, že je v aplikaci podporováno zadávaní nejen samostatných polynomiálních a exponenciálních funkcí, logaritmů, faktoriálů a konstant, ale také operace násobení, umocňování a sčítání.

Zadané argumenty faktoriálů a logaritmů jsou v implementaci gramatiky dále validovány. Při porušení pravidel gramatiky je uživatel informován o dané chybě chybovým oznámením v GUI aplikace, které obsahuje znázorněnou pozici neplatného znaku, popřípadě přesné znění chyby pro danou funkci.

4.3 Tvorba syntaktických stromů

Rekurzivní sestup bezkontextovou gramatikou tvoří pro každou z funkcí syntaktický strom. Přičemž rozlišuje 4 základní typy uzlů v tomto stromu. A to „Symbol“, což je v našem řešení vždy znak „n“ a „Konstanta“, tento typ reprezentuje libovolnou konstantu. Tyto první dva typy uzlu jsou vždy listy stromu. Dále uzel typu „Funkce“, která může být faktoriál (jeden potomek uzlu) nebo logaritmus (dva potomci uzlu) a v neposlední řadě pak uzel typu „Operátor“, který reprezentuje operace plus, násobení nebo umocňování a má vždy dva potomky.

Pro ukázku tvorby stromu nám poslouží objektová reprezentace vytvořeného stromu pro výraz „$33n^2 + ln(n) + 4n!$“ na obrázku 5



Obrázek 5: Příklad vytvořeného syntaktického stromu pro uživatelem zadanou funkci

4.4 Nalezení asymptoticky nejrychleji rostoucího prvku funkce

Pro nalezení asymptoticky nejrychleji rostoucího prvku funkce je potřeba rekurzivně projít dříve vytvořený syntaktický strom.

Asymptoticky nejrychleji rostoucí prvek pro každou z funkcí budeme reprezentovat ve formátu uspořádané trojice („x ,y, z“), kde „y“ značí hodnotu reprezentující daný prvek, „y“ jeho parametr a „z“ reprezentuje konstantu, kterou je násobena asymptoticky nejrychleji rostoucí funkce. Uchování konstanty „z“ má význam pro správné zpracování některých výrazů. Např. zápis Cpol, 7, 1 značí, že byl ve funkci nalezen polynom stupně sedm.

Pro nalezení asymptoticky nejrychleji rostoucího prvku pro danou funkci použijeme metodu, která bude rekurzivně volat sama sebe a po skončení tedy vrátí uspořádanou trojici, která bude odpovídat asymptoticky nejrychleji rostoucímu prvku dané funkce.

Při zpracování uzlu typu „Operátor“ je zjevné, že není možné pouze vrátit uspořádanou trojici jako u ostatních typů uzlů, ale je nutné nejprve rekurzivně zpracovat tyto potomky a následně zpracovat tyto dva výsledky do jediného, který bude odpovídat právě výsledku operace na těchto dvou potomcích.

Pro zpracování operací musíme tedy určit, které výrazy jsme schopni zpracovat a jak. Z logiky věci vyplývá, že u násobení budeme řešit několik případů, a sice násobení čehokoli konstantou, násobení dvou polynomů, nebo násobení polynomu logaritmem a naopak. Obdobně i v případě umocňování. V případě, že zadaná funkce obsahuje složitější výraz je další postup popsán v podkapitole 4.6.

Zpracování násobení je pro zjednodušení popsáno i tabulkou 3, která je zobrazena na konci této kapitoly. V tabulce můžeme vidět výsledek zpracování jednotlivých typů elementů, přičemž hodnota „–“ značí složitější prvek, který bude řešen později.

V případě, že parsovaný uzel symbolizuje sčítání, je nutné z rekurzivně získaných potomků vybrat ten, který roste asymptoticky rychleji. K tomuto využijeme jednoduchého porovnání hodnot uspořádaných trojic, které jsme získali zpracováním levého a pravého potomka operace sčítání.

4.5 Porovnání funkcí

Po rozparsování syntaktických stromů nám pro obě funkce vznikne jediná uspořádaná trojice, která tak bude odpovídat asymptoticky nejrychleji rostoucímu prvku funkce. Tyto prvky pak můžeme snadno porovnat pomocí hodnot trojic a tím získáme asymptoticky rychleji rostoucí funkci.

4.6 Porovnání funkcí pomocí Wolfram Alpha API

Při parsování stromu jsme narazili na výrazy, které není možno algoritmicky rozumně porovnat. Z vlastní iniciativy jsem zvolil další postup řešení pro takovéto případy, a to pomocí Wolfram API abych uživatelům nabídl co největší množinu funkcí, pro které si mohou porovnat jejich asymptotický růst. Zároveň jsem však z výše popsaných důvodu musel zachovat vlastní řešení porovnávání, aby uživatel při výpadku služby Wolfram (z libovolného důvodu) nebyl ochuzen o asymptotické porovnání růstu u funkcí, jejichž zadání umožňuje získat výsledek bez použití tohoto API.

Pro výpočet je zadána limita podílu zadaných funkcí jdoucí k nekonečnu ($\lim_{x\to\infty} \frac{f(n)}{g(n)}$). Na základě výsledku tohoto příkladu můžeme rozhodnout o tom, která ze zadaných funkcí roste asymptoticky rychleji. Tabulka 2 popisuje možné výsledky této operace.

Tabulka 2: Možné výsledky limity $\lim_{x\to\infty} \frac{f(n)}{g(n)}$ od Wolfram API

Výsledek	Význam při porovnání funkcí
infinity	Funkce f roste asymptoticky rychleji než funkce g, $g \in O(f)$
0	Funkce g roste asymptoticky rychleji než funkce f, $f \in O(g)$
konstanta	Zadané funkce rostou asymptoticky stejně rychle

V aplikaci je definována konstanta `wolframAPIKey`, která obsahuje hodnotu představující klíč pro přístup k Wolfram Alpha API. Tento klíč je pro nekomerční účely poskytován zdarma avšak s limitem 2000 požadavků na toto API za měsíc. Při překročení tohoto limitu nebo při jakékoli jiné chybě, která by eventuálně mohla nastat při spojení, a nebyl by tak zjištěn výsledek požadované limity, vrátí aplikace uživateli hlášku „Zadané funkce se nepodařilo asymptoticky porovnat. Funkce jsou příliš komplikované pro určení asymptotické rychlosti růstu a výpočet prostřednictvím Wolfram API se rovněž nezdařil.“. Pro budoucí použití není problém registrovat

placenou, neomezenou verzi tohoto API a jen změnit hodnotu konstanty `wolframAPIKey` na hodnotu nového klíče.

4.7 Vykreslení grafu

Mimo samotné funkční hodnoty funkcí bude pro vykreslení grafu potřeba další tři hodnoty, a to konečná a počáteční hodnota n (kde konečná hodnota musí být větší než hodnota počáteční, a jejich přednastavený rozsah je 0-5) určující rozsah grafu a hodnota konstanty, což je kladné reálné číslo s přednastavenou hodnotou 1 násobící asymptoticky rychleji rostoucí funkci. Tyto hodnoty může uživatel, podobně jako u zadávání funkcí, definovat vstupem z klávesnice i pomocí myši.

Pro výpočet funkčních hodnot grafu využijeme Math.js popsané v kapitole 6.5, která umožňuje práci s velkými čísly (nad limity běžných datových typů). V úvahu také musíme vzít nutnost zaokrouhlení argumentu faktoriálu.

Samotné vypočtení funkčních hodnot je závislé na předešlém porovnání funkcí, neboť konstantou je násobena funkce, která roste asymptoticky rychleji (pokud funkce rostou asymptoticky stejně rychle, je konstantou násobena funkce G).

Tabulka 3: Zpracování uzlu syntaktického stromu - násobení

Typ prvků / Hodnota	Konstanta / X	Logaritmus	Polynom / X	Pol. X * Log. / X	x^n / X	Faktoriál	n^n
Konstanta / Y	Konstanta / X*Y	Logaritmus	Polynom / X	Pol. X * Log. / X	x^n / X	Faktoriál	n^n
Logaritmus	Logaritmus	–	Pol. X * Log. / X	–	–	–	–
Polynom / Y	Polynom / Y	Pol. Y * Log. / Y	Polynom / X+Y	–	–	–	–
Pol. Y * Log. / Y	Pol. Y * Log. / Y	–	–	–	–	–	–
y^n / Y	y^n / Y	–	–	–	–	–	–
Faktoriál	Faktoriál	–	–	–	–	–	–
n^n	n^n	–	–	–	–	–	–

5 Návrh

Tato kapitola se věnuje návrhu systému, rozdělení aplikace na základní moduly a popisu uživatelského rozhraní.

5.1 Složení systému

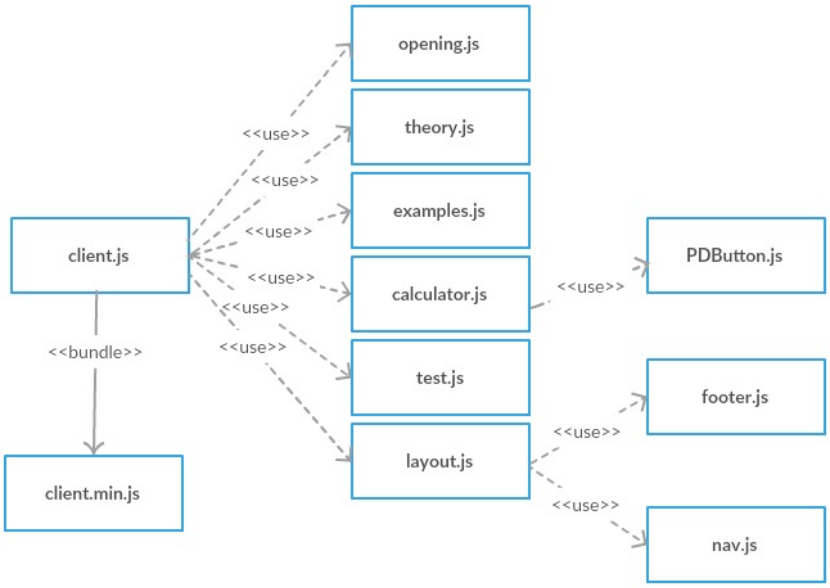
JavaScriptový kód aplikace je celý obsažen v jediném JavaScriptovém souboru `client.min.js` a tvoří tak jedinou komponentu technologie React (o technologii React se více zmíníme v kapitole 6.1). Tento soubor je pak načítán do HTML souboru `index.html`, kde společně s CSS soubory a externě načítaným MathJax.js souborem z internetu, který z technologických problémů nemůže být součástí souboru `client.min.js`, tvoří strukturu aplikace. Takto vzniklý krátký zdrojový kód HTML souboru `index.html` můžeme vidět ve výpisu programu 1.

```
<!DOCTYPE html>
<html lang="en">
   <head>
      <meta charset="utf-8">
      <meta http-equiv="X-UA-Compatible" content="IE=edge">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <meta name="description" content="asymptotic notation">
      <meta name="author" content="Martin Frydl">
      <title>Asymptoticka notace</title>
      <link rel="shortcut icon" href="images/favicon.ico" type="image/x-icon">
      <!-- Bootstrap CSS a vlastni CSS -->
      <link href="css/bootstrap.min.css" rel="stylesheet">
      <link href="css/bootstrap-theme.min.css" rel="stylesheet">
      <link href="css/styles-page.css" rel="stylesheet">
   </head>
   <body>
      <!-- Jadro aplikace nacitane z client.min.js -->
      <div id="app"></div>
      <!-- JavaScript -->
      <script type="text/javascript" src="client.min.js"></script>
      <script type="text/javascript" src="https://cdn.mathjax.org/mathjax/
         latest/MathJax.js?config=TeX-MML-AM_CHTML"></script>
   </body>
</html>
```

Výpis 1: Zdrojový kód kód HTML souboru index.html

5.2 Rozložení aplikace na moduly

JS soubor `client.min.js` je minimalizovaný (pomocí technologie Webpack jejíž základní popis je k dispozici v kapitole 6.7) zápis všech vytvořených JavaScriptových souborů a importovaných balíčků použitých při vývoji. Tento soubor vzniká na základě závislostí mezi vytvořenými JS soubory a jejich importy. Diagram popisující tyto závislosti a následně vznik tohoto souboru můžeme vidět na obrázku 6.



Obrázek 6: Diagram závislostí JS souborů

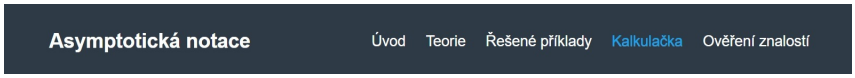
5.3 Uživatelské rozhraní

Uživatelské rozhraní webové aplikace je tvořeno komponenty a jejich vzhled je upravován pomocí vlastích CSS stylů v součinnosti s technologií Bootstrap (základní popis Bootstrap frameworku je k dispozici v kapitole 6.6). Při navrhování uživatelského rozhraní jsem se soustředil na dodržení zásad designu moderního webu, zejména na responzivitu, vizuální přitažlivost, celistvost barev a typografie, logické uspořádání komponent webu a přehlednost.

Nyní si popíšeme nejdůležitější ovládací prvky uživatelského rozhraní.

5.3.1 Menu

Menu je v aplikaci tvořeno klasickým horizontálním panelem s odkazy na jednotlivé stránky. Odkaz na aktivní stránku je pro přehlednost vyznačen modrou barvou. Ukázku můžeme vidět na obrázku 7.

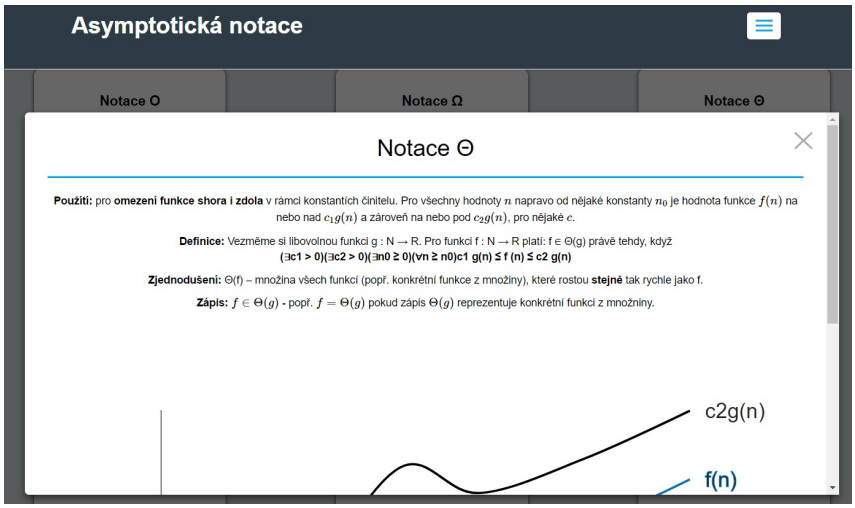


Obrázek 7: Responzivní menu aplikace

5.3.2 Teorie

Webová aplikace nabízí uživateli formou jedné webové stránky zopakovaní teorie k řešené problematice. Pro zajištění maximální přehlednosti pro uživatele je tento text rozdělen do kapitol.

Uživatel vybere kapitolu kliknutím na příslušný obdélník. Aplikace zobrazí obsah této kapitoly prostřednictvím zobrazení modálního okna. Uživatel může toto nově vzniklé okno zavřít vybráním jiné kapitoly, kliknutím mimo aktivní prvek nebo kliknutím na vykreslený křížek v pravém horním rohu aktivního prvku. Toto zpracování můžeme vidět na obrázku 8.



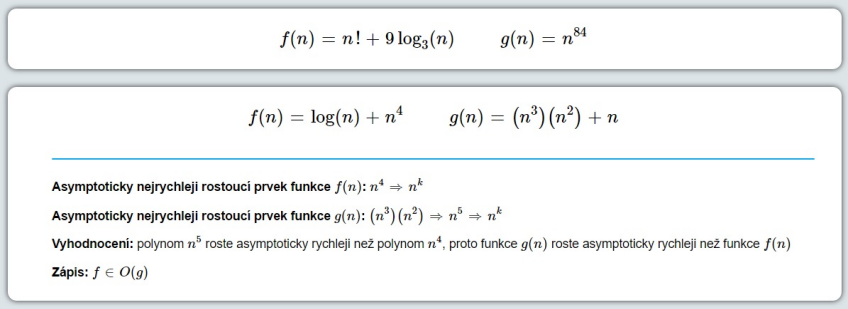
Obrázek 8: Ukázka zobrazení teorie v aplikaci

5.3.3 Řešené příklady

Na stránce s názvem **„Řešené příklady“** je uživateli nabídnuto několik základních řešených příkladů, které jsou určeny pro usnadnění pochopení teorie a principů sloužících k porovnávání funkcí pomocí asymptotické notace.

Při návrhu této stránky jsem se inspiroval stylem, který je dobře známý z různých internetových matematických fór. Uživateli je automaticky zobrazeno pouze zadání příkladu a jeho řešení uživatel zobrazí nebo skryje kliknutím na toto zadání.

Ukázku těchto příkladů z aplikace můžeme vidět na obrázku 9.

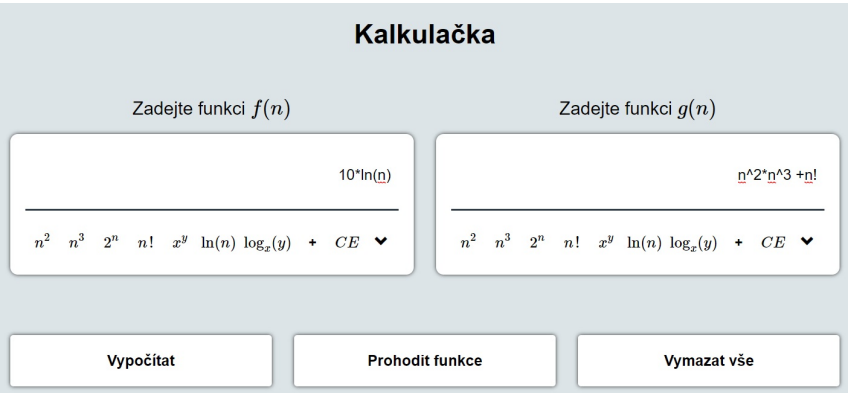


Obrázek 9: Řešené příklady

5.3.4 Kalkulačka

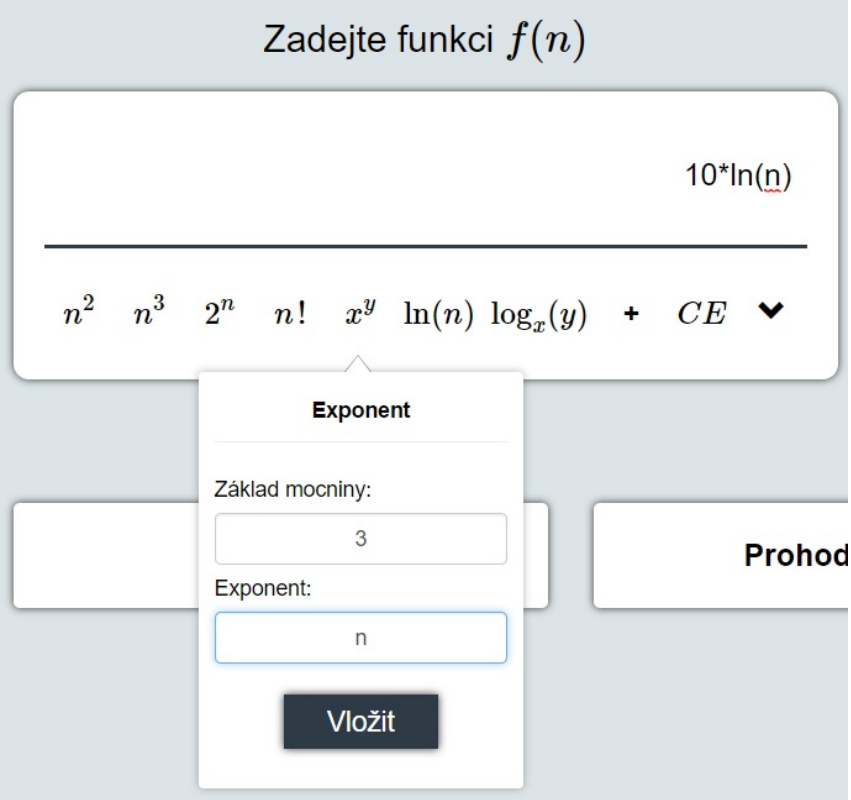
Na stránce webové aplikace obsahující kalkulačku si uživatel může zadat dvě vlastní funkce, které jsou porovnány pomocí asymptotické notace a následným výstupem je nejen textový výsledek oznamující, která ze zadaných funkci roste dle asymptotické notace v závislosti na počtu prvků rychleji, ale především graf znázorňující průběhy těchto funkcí.

5.3.4.1 Zadávání funkcí Uživatelské rozhraní pro zadávání funkcí můžeme vidět na obrázku 10. Z obrázku vyplývá, že uživatel má možnost definovat funkce nejen vstupem z klávesnice, ale pro usnadnění může využít kliknutí na jedno z tlačítek pod polem se zadávanou funkcí. Po kliknutí se hodnota výrazu na tlačítku automaticky dopíše na poslední známou pozici kurzoru v daném poli pro zadávání.



Obrázek 10: Rozhraní pro zadávání funkcí

Některá tlačítka vyžadují doplnění parametru pro zadaný výraz. Ukázku takového tlačítka můžeme vidět na obrázku 11



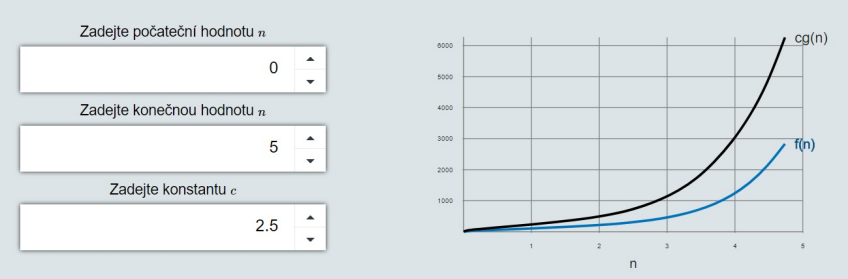
Obrázek 11: Ukázka tlačítka se zadáváním parametru výrazu

Aplikace také nabízí seznam předdefinovaných funkcí. Uživatel si může snadno vybrat z tohoto seznamu pomocí najetí kurzoru myši na tlačítko pod kalkulačkou se zobrazenou šipkou směřující dolů a následným kliknutím na konkrétní předdefinovanou funkci.

Seznam předdefinovaných funkcí je uložen v souboru `pre_defined_functions.json` a tlačítka znázorňující jednotlivé funkce jsou tedy generovány pomocí tohoto souboru při kompilaci. Toto řešení umožňuje snadnější úpravu aplikace při budoucím využití ve výuce.

Stiskem tlačítka **„Vypočítat“** provede uživatel samotný výpočet pro porovnávání zadaných funkcí. V případě korektních vstupů se zobrazí graf i textový výstup. V opačném případě je uživatel upozorněn na chybu rovněž textovým výstupem. Tlačítko **„Prohodit funkce“** provede záměnu obsahu polí obou funkcí a tlačítko **„Vymazat vše“** smaže obsah polí pro zadávání i případné vypočtené výstupy.

5.3.4.2 Graf Na obrázku 12 je graf znázorňující průběh zadaných funkcí. K dodatečnému ovládání tohoto grafu slouží tři textová pole pomocí nichž může uživatel nastavovat rozsah sledované oblasti grafu a konstantu pro násobení rychleji rostoucí funkce. Validita hodnot zadávaných v těchto polích je ověřována.

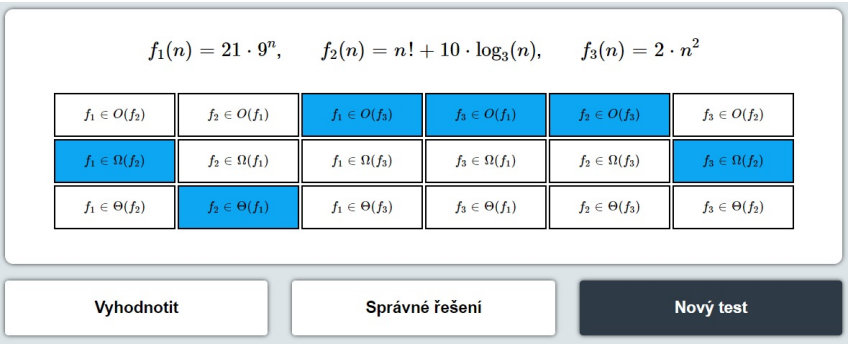


Obrázek 12: Graf znázorňující průběh zadaných funkcí

5.3.5 Testy

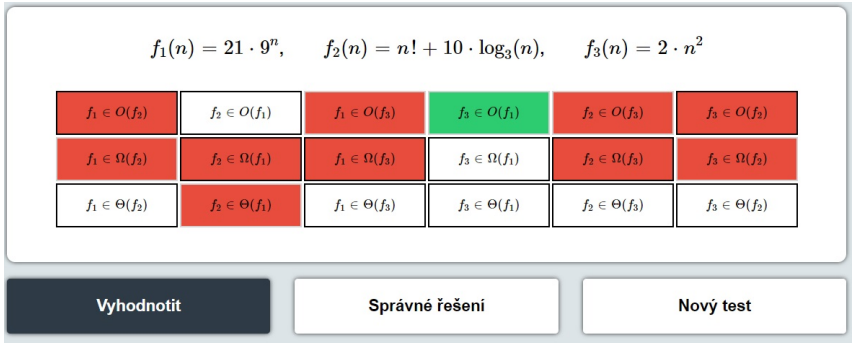
Na stránce s názvem **„Ověření znalostí“** má uživatel možnost vyplnit si, výše už několikrát zmiňované, náhodně generované testy na porovnání tří funkcí pomocí zápisů asymptotické notace. Podobný formát testů se může objevit u písemných zkoušek z předmětů zabývajících se touto oblastí a jejich vyplňování tak může být pro studenta velmi přínosné.

Test je vygenerován automaticky při příchodu na stránku nebo po stisku tlačítka **„Nový test“**. Uživatel kliknutím označí odpovědi, které považuje za správné. Označené odpovědi jsou označeny **modře**. Takto vyplněný test můžeme vidět na obrázku 13.



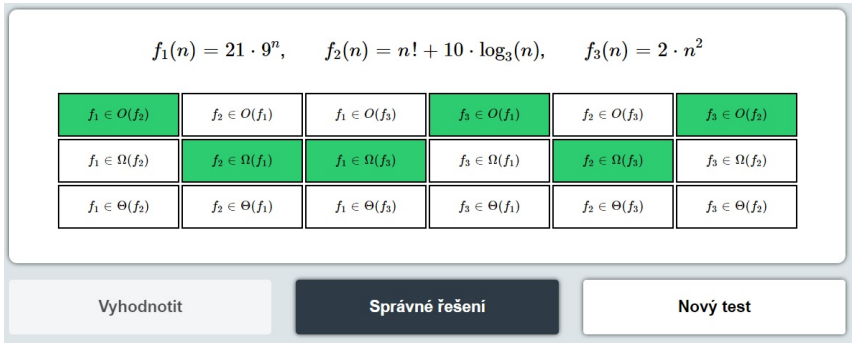
Obrázek 13: Vygenerovaný test s uživatelem označenými odpověďmi

Po stisku tlačítka **„Vyhodnotit“** systém vyhodnotí vyplněný test. Správné odpovědi označí **zelenou** barvou, chybné odpovědi a také správné, ale uživatelem neoznačené odpovědi pak barvou **červenou**. U odpovědí, které uživatel vybral a již byly hodnoceny, se změní barva ohraničení na **šedou**. Což uživateli usnadní orientaci pokud chce stejný test vyhodnotit opakovaně. Ukázku vyhodnoceného testu můžeme vidět na obrázku 14.



Obrázek 14: Vyhodnocený test

Uživatel má možnost zobrazit si pouze správné odpovědi kliknutím na tlačítko **„Správné odpovědi“**, jak můžeme vidět na obrázku 15.



Obrázek 15: Správné odpovědi v testu

6 Použité technologie

V této kapitole si představíme nejdůležitější technologie, frameworky a knihovny využity při vypracování této bakalářské práce a jejich samotné použití v systému.

Použité technologie jsem volil tak, aby byly multiplatformní a vzájemně na sebe navazovaly a doplňovaly se.

6.1 React

React [3] je JavaScriptová knihovna pro vytváření webových komponent. Dostupná je od roku 2013 kdy jí Facebook, který jí už několik let před tím sám interně používal a vylepšoval převedl pod Open Source licenci.

Nejzřejmější výhoda této knihovny je ta, že nás prakticky úplně odstíní od DOMu. V React komponentách pouze deklarativně zadefinujeme strukturu (HTML) skládáním JS funkcí. Na základě této struktury si React vytvoří vlastní virtuální DOM, který porovnává se skutečným DOMem a aktualizuje ho dle nalezených rozdílů.

Pomocí několika doplňkových knihoven, např. React Router, je umožněno vytvořit webovou aplikaci stylem „single page“, což je jednostránkový přístup k tvorbě webových aplikací, při kterém se mění pouze obsah jednotlivých stránek a není potřeba znovu načítat celou webovou stránku. A to za účelem dosažení větší rychlosti načítání jednotlivých stránek a co možná největší přehlednosti a pohodlí pro uživatele.

Použití této technologie v řešení bylo stěžejní. Pomocí knihovny React je vypracovaná jak struktura stránky tak i většina samotného zdrojového kódu.

6.2 jQuery

jQuery [4] je nejrozšířenější JavaScriptová knihovna s jednoduchou syntaxí. Klade důraz na interakci mezi JavaScriptem a HTML. Pro práci s DOMem webu používá selektory.

jQuery ve zdrojovém kódu slouží pro doplňkovou práci s již vyrenderovaným obsahem webu, převážně pro dynamickou úpravu CSS stylů přidělených elementům webu.

6.3 Victory

Victory [5] je knihovna pro modelování grafů a datovou vizualizaci, přizpůsobená pro použití jako komponenta technologie React.

Tato technologie je v aplikaci používaná pro vykreslování grafů a pro práci s nimi.

6.4 MathJax

MathJax [6] je JavaScriptová knihovna pro zobrazování matematických výrazů a rovnic ve webových prohlížečích za pomocí MathML.

Mathematical Markup Language (MathML) je značkovací poskytující grafickou prezentaci zadaných matematických výrazů při zachování smyslu výrazu.

MathJax knihovna je ve webové aplikaci, vzhledem k častému výskytů matematických zápisů, použita na všech stránkách. Mimo jiné je použita i pro překreslování zadávaných rovnic v kalkulačce do prezentovatelné podoby.

6.5 Math.js

Math.js [7] je matematická knihovna pro JavaScript s podporou pro velkou škálu různých výpočtů a práci s daty

Tato technologie je využita na stránce s kalkulačkou k výpočtům funkčních hodnot zadaných funkcí a převodu mezi formáty čísel pro vykreslení grafu.

6.6 Bootstrap

Bootstrap [8] je nejvíce populární HTML, CSS a JS framework poskytující sadu nástrojů mimo jiné pro práci s typografií a rozložením stránky usnadňující vývoj responzivních a multiplatformních webových aplikací.

Bootstrap framework je v systému využitý pro vytvoření responzivního uživatelského rozhraní.

6.7 Node.js, Webpack, npm manager

Tyto tři technologie jsou použity pro tvorbu aplikace jako celku. Pomocí **Node.js** [9], což je JavaScriptové prostředí s V8 JavaScript engine od společnosti Google, se instaluje **npm manager** [11] a následně konkretní **npm balíčky** potřebné pro tvorbu aplikace.

Z tohoto prostředí se rovněž spouští kompilace aplikace a zároveň bundlovací nástroj **Webpack** [10] jehož úkolem je všechny používané JavaScriptové soubory překompilovat do jediného JavaScriptového souboru neboli jediné komponenty knihovny React za účelem omezit počet HTTP požadavků a dosáhnutí vyšší rychlosti aplikace.

7 Implementace

Tato kapitola popisuje využití nejdůležitějších technologií použitých při implementaci a postupy implementace vedoucí k výsledné funkcionalitě aplikace.

Detailní popis implementace bude věnován pouze stránkám s kalkulačkou a generovanými testy, neboť ostatní stránky aplikace pracují pouze se statickým obsahem renderovaným pomocí HTML značek a CSS stylů.

7.1 React komponenty tvořící aplikaci

Kapitola 6.1 obsahuje zmínku použití technologie React při implementaci aplikace. Nyní si podrobněji představíme React komponenty, ze kterých se aplikace skládá. Pro ukázku nám poslouží základní struktura výsledného obsahu stránky.

V ukázce kódu 2 vidíme obsah zdrojového souboru `Layout.js`, který definuje rozložení stránky jako takové a rozděluje ji do tří základních komponent.

V horní polovině můžeme vidět import dvou dalších zdrojových souborů, resp. komponent a sice `Nav.js` a `Footer.js` neboli zdrojové kódy pro menu a obsah. Tyto komponenty jsou poté umístěny do struktury webu pomocí `<Footer/>` a `<Nav/>`.

Pomocí `const {location} = this.props;` definujeme aktivní podstránku webu a v obsahu stránky následně pomocí `this.props.children` přiřazujeme konkrétní komponentu tvořící podstránku webu do obsahu stránky.

Používáním komponent docílíme toho, že se stránka nemusí v průběhu procházení webu znovu načítat, ale pouze se vymění komponenta definující aktivní podstránku. Tímto postupem omezíme počet HTTP požadavků a dosáhneme vyšší rychlosti a zároveň uživateli nabídneme daleko vyšší uživatelský zážitek a pohodlí, které zná z desktopových aplikací.

```
import React from "react";
import { Link } from "react-router";

import Footer from "../components/layout/Footer";
import Nav from "../components/layout/Nav";

export default class Layout extends React.Component {
  render() {
    const { location } = this.props;
   const containerStyle = {
       marginTop: "60px",
    width: "100%",
    padding: "0px"
   };
```

```
    const pageStyle = {
      width: "100%"
    };

    return (
         <div style={pageStyle}>
            <Nav location={location} />
        <div className="container-fluid" style={containerStyle}>
              {this.props.children}
        </div>
        <Footer/>
      </div>
    );
  }
}
```

Výpis 2: Struktura webové stránky vytvořená pomocí React komponent

Výše zmíněné `props` jsou prostředek pro předávání dat mezi jednotlivými komponentami aplikace. Opakem je `state`, což je lokální stav komponenty, do kterého si můžeme ukládat libovolná data. Jeho hodnotu měníme pomocí metody `this.setState()`.

K zavolání metody `render()` dochází, pokud se změní `this.state` nebo `this.props`, případně při zavolání `forceUpdate()`.

K práci s komponentami jsou v aplikaci hojně využity i metody `componentDidMount()` a `componentDidUpdate()`, přičemž první zmíněná je volána pouze jedinkrát po prvním dokončení metody `render()` druhá naopak po každé takto provedené aktualizaci DOMu.

7.2 Implementace grafů

Jak už bylo zmíněno v kapitole 6.3 pro práci s grafy v této aplikaci je použita knihovna Victory a to především kvůli její optimalizaci pro technologii React. Hlavní výhodou je možnost definování hodnot a dalších vlastností grafů pomocí `this.state`. Při změně hodnot těchto stavů dojde k automatickému překreslení grafu.

7.3 Matematické zápisy

Matematické zápisy jsou v aplikaci řešeny pomocí knihovny MathJax popsané v kapitole 6.4. Text ve zdrojovém kódu ohraničený pomocí ‘ ‘ nebo `\( )\` např. `\(f = O(g)\)`, překreslí tato technologie ve výsledné aplikaci pomocí MathML do prezentovatelného matematického zápisu.

V aplikaci je často použita metoda `MathJax.Hub.Queue()`, která najde všechny ještě nepřekreslené matematické zápisy a postará se o jejich překreslení.

7.4 Kalkulačka

Dvě uživatelem zadané funkce jsou porovnány na základě jejich asymptotického růstu. Tato podkapitola popisuje implementaci zadávání těchto funkcí, jejich následné porovnání a vykreslení grafů s průběhy těchto funkcí.

7.4.1 Zadávání funkcí

Pro zadávání funkcí jsou použity ContentEditable prvky uzpůsobené pro použití v React.js. Obsah těchto prvků je nastaven pomocí lokálního stavu komponenty `calc.calcInputF` resp. `calc.calcInputG`. S tím jak se mění hodnota těchto stavů, mění se vlastní obsah ContentEditable prvků.

Každý z ContentEditable prvků má nastavenu „onChange“ událost, která zachytává změny v jeho obsahu, a při každé změně volá metodu `onInputChangeCalc(input, event)` kde parametr „input“ značí, která funkce je upravována. Tato metoda aktualizuje obsah daného ContentEditable prvku pomocí přidružených lokálních stavů.

Zmíněné metody umožňují samotný vstup z klávesnice. Avšak pro uživatele by mělo být primárním způsobem zadávání využití tlačítek s přednastavenými výrazy. Tlačítka mají nastavenu „onClick“ událost, která zachytává kliknutí na dotyčné tlačítko.

Uživatelem zadané bílé znaky jsou při výpočtu smazány. Zadávané funkce prochází validací pomocí bezkontextové gramatiky pro zajištění korektních výsledků. Popis a návrh gramatiky je obsažen v kapitole 4.2.

Pro ukázku z implementace této gramatiky pomocí analýzy rekurzivním sestupem můžeme vidět ve výpisu zdrojového kódu 3 implementaci přepisovacích pravidel neterminálu „Z“. Konkrétně vytvoření nového uzlu stromu představujícího logaritmus. Ze zápisu gramatiky vyplývá, že tento neterminál, spolu s neterminály „W“, „P“ a dalšími, slouží ke zpracování logaritmů.

```
parseZ(){
   ...
  if (calc.functionInput[calc.t] === "n") {
      return new math.expression.node.FunctionNode(new math.expression.node.
          SymbolNode('log'), [this.parseP(), ec]);
  }
   else if ...
}
```

Výpis 3: Vznik uzlu stromu představujícího logaritmus

7.4.2 Porovnání funkcí

Princip porovnávání funkcí vychází z analýzy v kapitole 4.

Uživatelem zadané funkce jsou porovnány na základě asymptotické rychlosti růstu. Pro toto porovnání je potřeba z obou zadaných funkcí určit asymptoticky nejrychleji rostoucí prvek. Tento prvek získáme na základě rekurzivního parsování syntaktického stromu, který se vytváří pomocí analýzy rekurzivním sestupem zadanou funkcí na základě bezkontextové gramatiky, jak je popsáno v předchozí kapitole 7.4.1.

Pro nalezení asymptoticky nejrychleji rostoucího prvku pro danou funkci je použita metoda `findBiggestElementFromTree(node, fce)`. Parametr „fce“ slouží pro informaci, která ze zadaných funkcí je právě zpracovávána a tedy nastavení správných lokálních stavů na proměnné. Tato metoda rekurzivně volá sebe samu a parametr „node“ reprezentuje uzel stromu, který bude zpracován. Při prvním volání metody tedy parametr „node“ představuje kořen syntaktického stromu dané funkce.

V ukázce kódu 4 můžeme vidět zpracování uzlů typů „ConstantNode“, tedy konstanty, jako část implementace metody `findBiggestElementFromTree(node, fce)`. V této ukázce můžeme vidět, že nově zpracovaný prvek je vždy vrácen jako uspořádaná trojice.

```
switch (node.type) {
   case 'ConstantNode':
      // Zpracování constanty
      return [Cconstant, 1, parseInt(node.value)]
      break;
      ...
```

Výpis 4: Zpracování konstant

Oproti výše uvedeným typům uzlů je 4. možný typ, „OperatorNode“ nutno dále rekurzivně zpracovat. Tento typ představuje operaci sčítání, násobení nebo umocňování. Uzel tohoto typu má tedy 2 potomky, pro které voláme metodu `findBiggestElementFromTree(node, fce)` za účelem jejich zpracování.

Metoda `findBiggestElementFromTree(node, fce)` umožňuje pracovat pouze s násobením „jednodušších“ prvků, a sice násobení čehokoli konstantou, násobení dvou polynomů nebo násobení polynomu logaritmem a naopak. Obdobně metoda pracuje i v případě umocňování. V případě, že zadaná funkce obsahuje složitější výraz uloží si tuto skutečnost prostřednictvím nastavení uspořádané trojice na „CTooCoplex“ („return [CTooComplex, -1, -1];“) a další zpracování je popsáno v textu níže.

V případě, že parsovaný uzel symbolizuje sčítání, je nutné z rekurzivně získaných potomků vybrat ten, který roste asymptoticky rychleji. K tomuto slouží metoda `compareNodes(e1, e2)`, která jako parametry přijímá právě tyto potomky (zpracované uspořádané trojice) a vrací hodnotu (hodnoty jsou opět uložené pomocí konstant na začátku zdrojového souboru) vyjadřující, který potomek roste asymptoticky rychleji, a to na základě porovnání hodnot. Nejprve porovnává typ prvku v případě shody pak hodnotu. Např. pro dvojici „[Cpol, 2]“ a „[Cpol, 3]“ vrátí

hodnotu pod konstantou „CmpFasterSecond“ značící, že druhý prvek „[Cpol, 3]“ roste rychleji. V opačném případě by výsledkem byla hodnota „CmpFasterFirst“, popřípadě pak „CmpEqual“ pro označení, že oba prvky rostou asymptoticky stejně rychle. Hodnotu „CmpTooHard“ pak vrací v případě, že alespoň jedna z uspořádaných dvojic představuje příliš složitý prvek.

Po zjištění asymptoticky nejrychleji rostoucího výrazu pro funkci F i G pomocí metody `findBiggestElementFromTree(node, fce)`, je nutno porovnat tyto výrazy mezi sebou, pro zjištění konečného výsledku pojednávajícím o tom, která ze zadaných funkcí roste asymptoticky rychleji. Toto porovnání je v aplikaci řešeno opět s použitím metody `compareNodes(e1, e2)`, která pomocí několika podmínek porovnává hodnoty „x ‚y“ z uspořádaný trojic „x ‚y, z“.

Uživateli je pak výstup zprostředkován pomocí metody `compareFunctions()`, která na základě porovnaných funkcí metodou `compareNodes(e1, e2)` vypisuje uživateli oznámení do GUI.

V případě, že metoda `findBiggestElementFromTree(node, fce)` nastaví při zmíněném problému s příliš složitými funkcemi uspořádanou trojici na „CTooComplex“, pak metoda `compareFunctions()` namísto výpisu volá metodu `compareFunctionsUsingWolfram()`. Tato metoda pro porovnání funkcí využívá externího Wolfram API, kterému je k výpočtu zadána limita podílu zadaných funkcí jdoucí k nekonečnu ($\lim_{x\to\infty} \frac{f(n)}{g(n)}$). Na základě výsledku tohoto příkladu můžeme rozhodnout o tom, která ze zadaných funkcí roste asymptoticky rychleji. Výpis zdrojového kódu 5 ukazuje jádro metody `compareFunctionsUsingWolfram()` odesílající požadavek na API pro vyřešení této limity. Možné výsledky této operace jsou popsány v kapitole 4.6.

```
var yqlURL = [
   "http://query.yahooapis.com/v1/public/yql",
  "?q=" + encodeURIComponent("select * from xml where url='" + xmlSource + "'")
      ,
  "&format=xml&callback=?"
].join("");

var _ = this;

$.getJSON(yqlURL, function(data){
var xmlContent = $(data.results[0]);
result = $(xmlContent).find("plaintext").text();
result = result.substring(0, result.indexOf("<"));
result = result.substring(result.indexOf("=")+2);
```

Výpis 5: Odeslání požadavku na výpočet limity

7.4.3 Vykreslení grafu

Způsob, jakým jsou v aplikaci vykreslovány grafy, je popsán v kapitole 7.2. Tato kapitola se věnuje popisu implementace grafu zobrazujícího uživatelem zadané funkce v kalkulačce.

Uživatelem zadané hodnoty pro práci s grafem procházejí validací pomocí několika metod, které na případně nekorektně zadané hodnoty upozorní uživatele výpisem chybového řádku do GUI.

Samotné vykreslení grafu je implementováno pomocí metody `drawGraph()`, která nejprve na základě uživatelem zadaného číselného rozsahu hodnot n určí hodnoty, pro které budou vypočteny funkční hodnoty rovnoměrným rozdělením tohoto rozsahu na konstantou určený počet hodnot.

Pro vykreslení grafu je nezbytné určit funkční hodnoty zadaných funkcí. Každá z těchto hodnot je určena dosazením vypočtené číselné hodnoty proměnné n do řetězce zadané funkce, a to pomocí metody `assignValuetoN(value)`. Následně se provede samotný výpočet funkční hodnoty pro právě dosazenou hodnotu proměnné n s použitím knihovny Math.js popsané v kapitole 6.5.

Metoda `assignValuetoN(value)` volá metodu `findFactorials()`. Tato metoda, jak její název napovídá, hledá pomocí práce s řetězci faktoriály a zaokrouhluje argument faktoriálu na celé číslo pro zajištění korektních výsledků.

Pro práci s velkými čísly je využit datový typ `BigNumber` knihovny Math.js popsané v kapitole 6.5. Ve výpisu zdrojového kódu 6 představující část implementace metody `drawGraph()` je zobrazena definice matematiky pro práci s velkými čísly a výpočet hodnot s nastavením popisků grafů funkcí ve vykresleném grafu pro případ, kdy funkce $f(n)$ je asymptoticky rychleji rostoucí nežli funkce $g(n)$.

```
math.config({
   number: 'BigNumber',
   precision: 64
});

if (calculate.functionsState === CmpFasterFirst) {
   graph.labelF = "cf(n)"
   graph.labelG = "g(n)"
   for (var i = 0; i < countOfGraphPoints; i++) {
      this.assignValuetoN(ticks[i]);
    valuesF.push(math.eval('(' + calculate.expressionF + ') * ' + constant.
       toString()));
    valuesG.push(math.eval(calculate.expressionG));
  }
```

```
}
```

Výpis 6: Výpočet funkčních hodnot pro vykreslení grafu

Rozsah grafu je nastaven dle uživatelem zadaného rozsahu pro proměnou n a vypočtených funkčních hodnot prostřednictvím metody `setChartDomain()`. V případě velmi vysokých funkčních hodnot je místo vykreslení grafu pouze zobrazena aktuálně nejvyšší hodnota pro obě funkce. Díky využití knihovny Math.js není maximální hodnota omezená, při extrémně vysokých hodnotách však dochází k navýšení výpočetního času a tím ke zpomalení aplikace.

7.5 Testy

Testy sloužící k ověření znalostí uživatele generuje samotná aplikace. Proto není nutné používat pro ukládání testových otázek žádnou formu databáze. Toto řešení má nesporné výhody nejen v různorodosti testů, ale také ve zjednodušení práce pro pedagogy, kteří tak nemusí sami tvořit a následně přidávat nové otázky do databáze.

Při generování funkcí v testu je použito několik metod pracujících s pravděpodobnostmi pro zajištění různorodosti těchto testů.

7.5.1 Generování funkcí

Pro generování funkcí je používáno pole představující hierarchii prvků pro asymptotické porovnávání funkcí, které je seřazeno od nejmenšího prvku po největší a je uloženo v lokálním stavu komponenty. Zápis zdrojového kódu definující toto pole můžeme vidět ve výpisu zdrojového kódu 7.

```
hierarchyOfFunctions: ['log(n)', 'n', 'n^k', 'k^n', 'n!', 'n^n']
```

Výpis 7: Pole představující hierarchii funkcí pro asymptotické porovnávání funkcí

Tuto hiearchii následně používají metody generující funkce pro porovnání. Základní metoda `genFunctions()` náhodně vybere 3 prvky z tohoto pole a jejich indexy představující jejich pozici v poli opět uloží jako lokální stav. Tyto vybrané prvky pak představují asymptoticky nejrychleji rostoucí prvky tří generovaných funkcí. Zdrojový kód metody `genFunctions()` je zobrazen jako výpis zdrojového kódu 8.

```
genFunctions(){
  const test = this.state.test;
  var x = 0;
  for(var i = 0; test.generatedFunctions.length < 3; i++){
    test.generatedFunctions.push(Math.floor(Math.random() * (test.
        hierarchyOfFunctions.length)));
```

```
  }
  this.setState({test: test});
}
```

Výpis 8: Zdrojový kód metody generující základní prvky funkcí

Asymptotické porovnání jediného výrazu by ale bylo pro uživatele příliš jednoduché. Proto se pro každou generovanou funkci volá i metoda `genSideOfFunction(index)`, která vrací hodnotu určující zdali, popř. na kterou stranu, se má ke generované funkcí přidat další výraz. Pro zajištění různorodosti generovaných funkcí pracuje tato funkce s určitými pravděpodobnostmi pro generování této návratové hodnoty. Procentuální vyjádření těchto pravděpodobností je zobrazeno v tabulce 4.

Tabulka 4: Návratové hodnoty metody genSideOfFunction(index)

Pravděpodobnost	Hodnota	Význam
20%	1	Levá strana
20%	2	Pravá strana
10%	3	Obě strany
50%	0	Nebude přidán další prvek

Na základě návratové hodnoty metody `genSideOfFunction(index)` je pak volána metoda `genNewFunction(index)`, která zajišťuje samotné vygenerování dalšího prvku z hierarchie.

Parametr metod `genSideOfFunction(index)` a `genNewFunction(index)` má v obou metodách velmi podobný význam. Souhrnně je jeho účelem zajistit, že nově vygenerovaný prvek z hierarchie funkcí bude mít asymptoticky pomalejší růst nežli původní prvek a zůstane tak zachována dominanta původního prvku v takto vygenerované funkci. Tímto způsobem je zajištěna 100% správnost testu, neboť odpadá nutnost složitějšího porovnávání vygenerovaných funkcí.

7.5.2 Generování konstant

Pro generování konstant, které doplňují generované funkce je použito několik metod, kde stěžejní je metoda `genConstant()`, která vrací náhodnou hodnotu následně použitou jako konstanta. Podobně jako metoda `genSideOfFunction(index)` i tato metoda pracuje určitými pravděpodobnostmi pro generování její návratové hodnoty. Procentuální vyjádření těchto pravděpodobností je zobrazeno v tabulce 5.

Tabulka 5: Návratové hodnoty metody genConstant()

Pravděpodobnost	Interval vygenerované konstanty
65%	2 - 10
25%	11 - 80
10%	81 - 100

Metoda `genConstant()` je využívaná například v metodě `replaceConstant(fce)`, která nahrazuje „k“ ve výrazech jí předaných jako parametr, a to pro zápisy „k^n“ a „n^k“ představujících prvky z hierarchie funkcí. Zejména pro zápis „n^k“ představující polynom je její použití důležité. Je spojeno s uložením této konstanty neboť u polynomů záleží na jeho stupni při následném asymptotickém porovnání. Stejně tak záleží na hodnotě exponenciální funkce „k^n“.

Další metodou pracující s pravděpodobností je metoda `shouldAddConstant()`, která vrací hodnotu typu „boolean“ jejíž hodnota je „true“ s pravděpodobností 30%. Tato metoda je použita při generování konstant před výrazy vybírané z hierarchie a pro přidávání hodnoty základu u logaritmů, jelikož není žádoucí aby každá generovaná funkce byla násobená konstantou nebo aby každý zápis logaritmu obsahoval základ logaritmu.

7.5.3 Vyhodnocení testu

Pro vyhodnocení odpovědí v testu je nejprve nutné určit správné odpovědi. To zajišťuje metoda `setCorrect()`, která porovnává indexy vygenerovaných dominantních prvků funkce popř. stupně polynomů nebo hodnoty exponenciálních funkcí. Na základě tohoto porovnání pak ukládá hodnoty do pole, které je uloženo jako lokální stav komponenty a ukládané hodnoty představují pořadí buňky v tabulce s odpověďmi v GUI.

Pro samotné vyhodnocení testu je použita metoda `checkAnswers()`, která používá pole hodnot uložené metodou `setCorrect()`. Na základě těchto hodnot upravuje CSS třídy konkretních buněk v tabulce s odpověďmi v GUI. Výsledkem je vyhodnocený test.

Podobně jako metoda `checkAnswers()` je implementována i metoda `showCorrect()` jejímž úkolem je zobrazit pouze správné odpovědi.

8 Nasazení

Vzhledem k použitým technologiím a způsobu implementace řešení funguje tato webová aplikace plně na straně klienta bez potřeb serverových operací nebo databázových systémů. Proto může být aplikace nasazena na libovolný webhosting a to včetně těch nejzákladnějších poskytovaných zdarma.

V průběhu vývoje byla aplikace testována a pravidelně prezentována vedoucímu práce prostřednictvím serveru HomeL. Tedy konkrétně na adrese http://homel.vsb.cz/ fry0050/bp/.

Pro nasazení aplikace tedy stačí pouze zdrojové soubory zkopírované ze složky „Deploy“ obsažené v příloze A do příslušného adresáře webového serveru. Pro zachování estetické kvality webové aplikace doporučuji využití služeb webhostingu neobsahujícího reklamy.

9 Závěr

Cílem této bakalářské práce bylo vytvořit komponentu výukového serveru pro podporu výuky teoretické informatiky, pomocí které si studenti budou moci prohloubit znalosti v oblasti Asymptotické notace.

Aplikace obsahuje přehledně zpracovanou teorii, kterou si tak studenti mohou snadno zopakovat kdykoli je to při práci s aplikací i mimo ní potřeba. Přiloženy jsou i řešené příklady s postupem výpočtů, které usnadňují pochopení principů porovnávání funkcí pomocí této notace. Dále pak aplikace nabízí vlastní kalkulačku pro porovnání uživatelem zadaných funkcí s textovým a grafovým výstupem. Sledovanou oblast grafu a konstantu pro výpočet hodnot pak uživatelé mohou za chodu dále obměňovat a pozorovat průběh růstu zadaných funkcí. Aplikace také poskytuje možnost ověření znalostí uživatele pomocí automaticky generovaných testů na téma porovnávání funkcí pomocí asymptotické notace.

Celé řešení je koncipováno tak, aby použité technologie byly co nejméně platformně závislé. GUI aplikace je postaveno na principech moderního webdesignu přičemž systém klade důraz na vizuální příjemnost aplikace, rychlost, přehlednost a jiné zásady, které uživatelé znají z desktopových aplikací.

Řešení je možné snadno rozšířit o další webové stránky nebo funkce pomocí technologií popsaných v této práci nebo s použitím základních znalostí HTML a JS přidat k aplikaci samostatně definované webové stránky. Nicméně aplikace je kompletní, samostatný a funkční celek a věřím, že najde své uplatnění při studiu problematiky asymptotické notace.

I přes velké množství času, které jsem touto bakalářské práci strávil, hodnotím práci velmi pozitivně. A to zejména z důvodu rozšíření mých znalostí z oblasti teoretické informatiky, implementace webových aplikací a také seznámení se s technologií React. Jmenovitě pak práce s bezkontextovou gramatikou při zadávání funkcí a také samotné téma bakalářské práce jsou zkušenosti, které určitě využiji při dalším studiu i v zaměstnání.

Literatura

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein *Introduction to Algorithms,*, 2nd ed. Cambridge, Mass.: MIT Press, 2001. ISBN 0-262-03293-7.

[2] Jan Neckář, *Algoritmy.net, Teorie algoritmů - Asymptotická složitost*, [online], 2016 [cit. 2017-03-10]. Dostupné z: https://www.algoritmy.net/article/102/Asymptoticka-slozitost.

[3] Facebook - React, *A JavaScript library for building user interfaces*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://facebook.github.io/react.

[4] jQuery, *Write less, do more*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://jquery.com/.

[5] Victory, *React.js components for modular charting and data visualization*, [online], 2017 [cit. 2017-04-12]. Dostupné z: http://formidable.com/open-source/victory/.

[6] MathJax, *Beautiful math in all browsers*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://www.mathjax.org/.

[7] Math.js, *An extensive math library for JavaScript and Node.js*, [online], 2017 [cit. 2017-04-12]. Dostupné z: http://mathjs.org/.

[8] Bootstrap, *Framework for developing responsive, mobile first projects on the web*, [online], 2017 [cit. 2017-04-12. Dostupné z: http://getbootstrap.com/.

[9] Node.js, *JavaScript runtime built on Chrome's V8 JavaScript engine*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://nodejs.org/en/.

[10] Webpack, *Module bundler*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://webpack.github.io/.

[11] npm, *Build amazing things*, [online], 2017 [cit. 2017-04-12]. Dostupné z: https://www.npmjs.com/.

A Příloha na CD

- **BP_FRY0050** zdrojové soubory aplikace
- **Deploy** - soubory pro nasazení aplikace
- **TeX** - zdrojové soubory LaTeXpro vysázení tohoto textu včetně použitých obrázků
- **Text_BP_FRY0050** - tento text bakalářské práce ve formátu PDF